User Data Analytics and Recommender System for Discovery Engine

Yu Wang

Master of Science Thesis
Stockholm, Sweden 2013

TRITA-ICT-EX-2013: 88
User Data Analytics and Recommender System for Discovery Engine

Yu Wang
yuwan@kth.se

Whaam AB, Stockholm, Sweden
Royal Institute of Technology, Stockholm, Sweden

June 11, 2013

Supervisor: Yi Fu, Whaam AB
Examiner: Prof. Johan Montelius, KTH
Abstract

On social bookmarking website, besides saving, organizing and sharing web pages, users can also discovery new web pages by browsing other's bookmarks. However, as more and more contents are added, it is hard for users to find interesting or related web pages or other users who share the same interests. In order to make bookmarks discoverable and build a discovery engine, sophisticated user data analytic methods and recommender system are needed.

This thesis addresses the topic by designing and implementing a prototype of a recommender system for recommending users, links and linklists. Users and linklists recommendation is calculated by content-based method, which analyzes the tags cosine similarity. Links recommendation is calculated by linklist based collaborative filtering method. The recommender system contains offline and online subsystem. Offline subsystem calculates data statistics and provides recommendation candidates while online subsystem filters the candidates and returns to users.

The experiments show that in social bookmark service like Whaam, tag based cosine similarity method improves the mean average precision by 45% compared to traditional collaborative method for user and linklist recommendation. For link recommendation, the linklist based collaborative filtering method increase the mean average precision by 39% compared to the user based collaborative filtering method.
Acknowledgement

I would like to express my sincere gratitude to ...

Yi Fu, CTO at Whaam
   for being my supervisor and helping me on the project

Johan Montelius, Professor at KTH
   for being my examiner and giving valuable suggestions to my
   report and presentation

All colleagues at Whaam
   for helping me in the company

My family
   for supporting me the many years of studies
1. Introduction

1.1 Whaam

Whaam [1] is a Stockholm based startup, currently building a social bookmark service and discovery engine. On Whaam, users can save their favorites links and store them neatly in one place. To keep links organized, users can create a linklist which groups similar links together and give each link tags and a category. By discovery, users can browser links or linklists by categories and tags, and follow other users who share the same interests to get inspiration.

In addition, users can search contents on Whaam by keywords. Whaam is also integrated with other social network like Facebook and Twitter, so users can share contents to other platform as well. Whaam wants to make discovery easy and intuitive even as the Internet keeps growing.

1.2 Discovery Engine and Recommender System

The number of websites has grown from 3 million to 555 million in last decade [2]. As many websites and webpages are added every day, it is hard for user to find interesting contents from Internet. One way to solve this problem is to use search engine, when you know what you are looking for. Users also like to use bookmark services like delicious [3] to save favorite links and recommend links to their friends. As social network websites like Facebook, Google+ become popular these days, users find interesting links through their friends. However, SNS only help users be connected and users can know what their friends are doing through feed. Users cannot really discover links based on their interests.

Discovery engine is another form of search engine, in which you don’t know what to search. Instead, you just want to find something interesting. Discovery engine knows your preferences, analyzes your history and recommends contents to you based on your interests. The essential part of a discovery engine is the recommender system, which makes the recommendation based on data. By definition, recommender systems are a subclass of information filtering system that seek to predict the ‘rating’ or ‘preference’ that a user would give to an item or social element they had not yet considered, using a model built from the characteristics of an item or the user's social
environment [4]. By combing the recommender system and the social bookmark service, we can build a discovery engine for the users.

1.3 Problem Description

Normally, user discovers interesting contents such as users and links by browsing the categories pages or by recommendation from their friends or followings. As more and more contents are added to the system, it is not easy for user to discover interesting contents. The three discover methods in current system have different problems as the size of contents grows. For discover by browsing the categories pages, the categories in Whaam are only general categories such as Sport, Fashion and etc. In each category, there are too many links or linklists and user will be overwhelmed by so many contents. For discover by recommendation from their friends, not all the user’s friends share the same interests, so the recommendation from friends is not always accurate. For discover by recommendation from user’s followings, as the size of users grows, the user still get overwhelmed and it is hard for user to find new followings. So as the data size grows, a discovery engine is needed and it can recommend contents to users accurately and efficiently.

In general, the problems in the current Whaam service are
- Only discovering in category page and friend feed is not enough
- Too many links added and too many users followed makes the user hard to find interesting staff

1.4 Methodology

We have three phases to solve the problem and prove the results. The first phase is the data analysis and algorithms design. In this phase, we analyze the data used in current Whaam system and propose algorithms for recommender system. The second phase is the prototype design and implementation. In this phase, we design and implement a prototype to evaluate the algorithm in the first phase. The last phase is the evaluation. In this phase, we use offline evaluation to measure the accuracy of the recommender system and algorithms. Based on the result of three phases. We make conclusion of our recommender system.

1.5 Contributions

We analyze the data structure and relations in Whaam and propose suitable algorithms for the recommender system, which can be used in social bookmark service like Whaam. We use different algorithms
for different recommendations. For user and linklist recommendation, we use tag based cosine similarity recommendation algorithm, which we think it is better than the traditional collaborative filtering method in the social bookmark area. This method is a kind of content-based recommendation algorithms. We use the tag frequency inverse tag global usage to represent user or linklist. Then the similarity of user or linklist is calculated using cosine similarity method. For link recommendation, we apply the traditional collaborative filtering method on linklist, which gives better result than the traditional item-to-item collaborative filtering method based on user. By using the similarity data we get from user and linklist recommendation, we can solve the sparsity problem in traditional collaborative filtering method. When one link is saved only by a small number of users, we can increase the number by including all the similar users.

In order to evaluate the algorithms and prove the feasibility of the recommender system, we also design and implement a prototype of recommender system based on social bookmark service for discovery engine, which can recommend users, links and linklists to users. The recommender system has offline subsystem and online subsystem. The offline subsystem calculates the data statistics and similarity while the online subsystem fetch the data from offline subsystem and responds to user’s request on real time.

We evaluate our prototype using part of live data in Whaam. The results show that tag based cosine similarity method improves the mean average precision by 45% compared to traditional collaborative method for user and linklist recommendation. For link recommendation, the linklist based collaborative filtering method increase the mean average precision by 39% compared to the user based collaborative filtering method.

1.6 Structure of the report

We begin with some background and related work (Section 2). From Section 2.1 to 2.3, we discuss three recommend algorithms: Content-based filtering, Collaborative filtering and Hybrid method. In section 2.4, we list two production recommender systems and the algorithm used. Section 3 is the main part, in which we analyze the problem and give the recommender system prototype design and implementation. We evaluate the system and show the result in section 4. A conclusion of our work is in section 5.
2. Backgrounds and Related Work

In this section, we first list three approaches to recommendations: content-based filtering, collaborative filtering and hybrid method. Then we analyze two popular recommender systems on the web.

2.1 Content-based Filtering

Content-based filtering method recommends items to users based on the representation of the items, which match the profile or interests of the users. The representation of items can be different forms or from different sources [5]. For example, links can have title, description, text contents, tags, categories etc. Title, description and text contents are from the link itself, while tags and categories may from users who save the link. The profiles of users can be set explicitly when they fill an interest forms, or implicitly by analyzing user’s old activities on the system or from users’ feedback about the items [6]. For unrestricted text in items, we can use the approaches in information retrieval system like calculating the term frequency times inverse document frequency [7] to get a list of keywords about the item. Content based filtering method needs enough data to start the analysis and sometimes it is hard to get representation of items.

The key in content-based filtering method is to match the item representation to user profile or interests representation. For example, in table 2.1, we have four articles and one user. They all use weighted keywords to represent themselves.

<table>
<thead>
<tr>
<th>Article</th>
<th>keyword1-&gt;10, keyword2-&gt;8, keyword3-&gt;5, keyword4-&gt;3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Article2</td>
<td>keyword3-&gt;12, keyword1-&gt;8, keyword2-&gt;5, keyword5-&gt;3</td>
</tr>
<tr>
<td>Article3</td>
<td>keyword5-&gt;9, keyword1-&gt;8, keyword6-&gt;5, keyword7-&gt;3</td>
</tr>
<tr>
<td>Article4</td>
<td>keyword4-&gt;7, keyword6-&gt;5, keyword8-&gt;2</td>
</tr>
<tr>
<td>User</td>
<td>keyword7-&gt;7, keyword1-&gt;5, keyword3-&gt;4, keyword5-&gt;3, keyword2-&gt;2</td>
</tr>
</tbody>
</table>

Table 2.1 content-based filtering example data

The keywords weights are not in the same scale. For example, article1 has weight from 10 to 3 while article2 has weight from 12 to 3. The first step is to make them in the same scale and use vector to represent them. One simple method is to make all weight divided by the maximum weight, so they all have the sample scale from 0 to 1. In addition, in order to make the vector the same size, we need to add the missing keyword with weight 0. There are totally eight keywords, so we have these vectors to represent the articles and user:
Then we can use the vector cosine similarity to calculate each article's similarity to the user. The equation of vector cosine similarity is:

$$\cos(V1, V2) = \frac{v1 \cdot v2}{|v1| \cdot |v2|}$$

Then we have:

- Cos (Article1, User) = 0.602
- Cos (Article2, User) = 0.678
- Cos (Article3, User) = 0.648
- Cos (Article4, User) = 0.0

From the result, we can recommend article2, then article 3, and then article 1 to the user. In addition, we should never recommend article 4 to the user.

### 2.2 Collaborative Filtering

Collaborative filtering is one of the most successful recommender techniques. The fundamental assumption of collaborative filtering method is that if users X and Y rate n items similarly, or have similar behaviors (e.g., liking, saving, listening), and hence will rate or act on other items similarly [8]. Collaborative filtering based recommendations can be divided into three groups: memory-based collaborative filtering techniques such as the neighborhood-based collaborative filtering algorithms and model-based collaborative filtering techniques such as Bayesian belief nets collaborative filtering algorithms, clustering collaborative filtering algorithms, and Markov decision processes based collaborative filtering algorithms, and hybrid collaborative filtering techniques such as the content-boosted collaborative filtering algorithms and Personality diagnosis [9].

The memory based algorithm uses users’ previous ratings to calculate the similarity of users or items to make prediction and the prediction can be based on similarity of user-item relationship or item-item relationship. The user-item relationship method recommends items based on what the similar users have rated. While
the item-item relationship approach recommends items based on other items, which users also rated at the same time, like "People who bought this item also bought ...". Memory based algorithm is effective and easy to implement and Amazon uses this kind of method [10].

For example, in Table 2.2, we have three users rating on three items. The rating is from 0 to 5 for all items. Here we want to predict the user2's rating to item2. We can use either user-item relationship or item-item relationship to predict the rating.

<table>
<thead>
<tr>
<th>User</th>
<th>Item1</th>
<th>Item2</th>
<th>Item3</th>
</tr>
</thead>
<tbody>
<tr>
<td>User1</td>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>User2</td>
<td>4</td>
<td>?</td>
<td>5</td>
</tr>
<tr>
<td>User3</td>
<td>3</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 2.2 collaborative filtering method sample data

For user-item relationship, we first calculate the similarity between users using vector cosine similarity.

\[
\text{Cos (User1, user2)} = \frac{(5,3) \cdot (4,5)}{|5,3| \cdot |4,5|} = 0.937
\]

\[
\text{Cos (User2, user3)} = \frac{(4,5) \cdot (3,4)}{|4,5| \cdot |3,4|} = 0.999
\]

Then the rating for item2 of user2 is:

\[
\frac{0.937 \times 3 + 0.999 \times 5}{0.937 + 0.999} = 4.032
\]

For item-item relationship, we first calculate the similarity between items using vector cosine similarity.

\[
\text{Cos (Item1, Item2)} = \frac{(5,3) \cdot (3,5)}{|5,3| \cdot |3,5|} = 0.882
\]

\[
\text{Cos (Item2, Item3)} = \frac{(3,5) \cdot (3,4)}{|3,5| \cdot |3,4|} = 0.995
\]

Then the rating for item2 of user2 is:

\[
\frac{0.882 \times 4 + 0.995 \times 5}{0.882 + 0.995} = 4.530
\]

The item-item relationship method can also be used to recommend similar items based on current item, which is used by Amazon.com [10] to show "Customers who bought this item also bought ...".
The model-based algorithm usually applies machine learning or data mining techniques in recommendation to learn a model from previous user’s data or pattern to make prediction. The model based algorithm method is more resilient to cold start and data sparsity problem [11][12]. However, it takes significant time to generate the model.

2.3 Hybrid Method

The hybrid method combines the content based filtering and collaborative filtering methods to make predictions or recommendations. The combination can avoid limitations of either content based and collaborative filtering methods and improve the performance. Different combinations exist like adding content based characteristics to collaborative filtering models, adding collaborative filtering characteristics to content based models or combining the result of both content based and collaborative filtering methods.

2.4 The Recommender System

A well-known application of recommender system is Amazon.com [13]. It uses item-to-item collaborative filtering method to recommend [10]. For example, on the item page, Amazon recommends related items by showing “Customers who bought this item also bought ...”:

Figure 2.1: Items recommendation on item page from Amazon.com

Product catalogs are also considered during recommendation so that only items within a product catalogs are shown in the recommendation list. For scalability, Amazon.com uses offline computation to calculate items similarity table, which makes online recommendation fast and the product catalogs cluster ease the similarity computation since the number of items within one product catalog is much smaller than the whole items.

The world’s most popular online video website Youtube.com [14] also uses recommender system to recommend videos. The system personalized sets of videos to users based on their activity on the site.
For example, on the recommended feed page, YouTube shows a list of videos based on what you watched before:

![Recommended videos for you](image1)

![Recommended videos for you](image2)

**Figure 2.2: “Recommended videos for you” on YouTube**

YouTube uses collaborative filtering to construct a mapping from video to a set of similar or related videos. Then recommendation candidates are generated from similar videos of user’s history. After that, recommended videos are ranked based on video quality, user interests and diversification. Since videos on YouTube have a short life cycle, the recommendation candidates are usually calculated from videos for a given time period (usually 24 hours). For scalability, YouTube also uses a so-called “batch-oriented pre-computation approach” rather than on-demand calculation. The data is updated several times per day.
3. User Data Analytics and Recommender System

In this section, we first analyze the problem and requirements. Then we propose the algorithms used for the recommender system. The design and development details for a prototype of the recommender system are given in the following chapters.

3.1 Problems and Requirement Analysis

In this section, based on the domain knowledge and requirement, we analyze the data structure used in current Whaam system and propose different recommendation methods for different data types.

3.1.1 Understanding the Domain Knowledge

Before analyze the problem and design the system, we need to understand the domain knowledge of current system and the purposes of our recommender system.

The main activities users can do on the system:
1. Register
2. Login
3. Change setting
4. Follow another user
5. Save link to a linklist
6. Like a link or linklist
7. Subscribe another user’s linklist
8. View a link or linklist
9. Comment on link and linklist
10. Share a link or linklist on other social platform

The data relations in Whaam are listed in Figure 3.1. One user has one profile and several followers. One user has several links and linklists. Links are grouped together to linklist. One link has several tags and linklist tags contain tags of each link in the linklist.
The goal of Whaam is making discovery easy and intuitive even as the Internet keeps growing. The recommend system can help to achieve the goal by recommending users to users who share the same interests and recommending links/linklists to users directly. Users can not only discover contents directly based on their profile but also discover other interesting staff from other users. After understanding the Whaam data relations and the goal of the discovery engine, we can list the requirements of the discovery engine for Whaam.

### 3.1.2 Functional Requirements and Non-functional Requirements

**Functional requirements**

1) Recommend users to user  
   For each user, a list of similar users is recommended. The similar users also exclude current user's followers because it is kind of repetition information.

2) Recommend links to user  
   Based on links user already saved, similar links are recommended to user.

3) Recommend linklists to user  
   Based on linklists that user already created and subscribed, similar linklists are recommended to user.

4) Show similar links/linklists  
   For each link/linklist page, similar links/linklists are recommended to user who views the link/linklist.

5) Provide REST service to web server to get the recommendation
The recommender system provides service to web server. The result is returned as a JSON response.

**Non-functional requirements**
1) Recommendation should be accurate.
2) Recommendation should be real time.
3) Recommendation can scale to millions of users and links.

### 3.1.3 User Activities and Data Analysis

In this section, we list all the activities that are needed to consider, the generated data and how to pre-process it before we do the recommendation. We find which recommendation is suitable for each data. In Whaam, all data is stored in MySQL database. The main tables and their attributes are listed in Table 3.1

<table>
<thead>
<tr>
<th>Type</th>
<th>Useful data attributes</th>
<th>Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Follows</td>
<td>id, user_id, follower_id</td>
<td>Collaborative filtering</td>
</tr>
<tr>
<td>User Profile</td>
<td>id, user_id, interests</td>
<td>Content based</td>
</tr>
<tr>
<td>Link</td>
<td>id, url, title, description, tags</td>
<td>Content based / collaborative filtering</td>
</tr>
<tr>
<td>Linklist</td>
<td>id, title, description, save_ids</td>
<td>Content based / collaborative filtering</td>
</tr>
<tr>
<td>Tag</td>
<td>id, name</td>
<td>Content based</td>
</tr>
<tr>
<td>Saves</td>
<td>id, link_id</td>
<td>Collaborative filtering</td>
</tr>
<tr>
<td>Likes</td>
<td>id, user_id, type, obj_id</td>
<td>Collaborative filtering</td>
</tr>
<tr>
<td>Subscribes</td>
<td>id, user_id, type, obj_id</td>
<td>Collaborative filtering</td>
</tr>
</tbody>
</table>

Table 3.1: Data type and suitable recommendation approaches

Not all the activities can contribute to the recommender system. Some activities out weight other activities. So when building the user profile model, we need to consider the priority of different kinds of activities. The data generated in the activities will have different weight for the input of the recommender system. For links recommendation, user activities about saving and liking links should have more priority than the viewing and commenting links; because from viewing or commenting, it is hard to know whether user likes the link or not and we will skip the viewing and commending activity data. Saving links should have more priority than liking links;
because saving link means this link is useful for the user however liking a link is just an attitude about the link. So when recommend links to user, the useful links should be shown before the links user may like. We use the same strategy to the linklist recommendation that creating and subscribing linklists have more priority than viewing and commenting linklists. For user recommendation, the links and linklists data contribute equally. Another activity is sharing link or linklist. We will also skip these activities because users always share which they have already saved or liked and it may be redundant as the saving or liking activity. Therefore, we only considering save, like and subscribe activities here.

Links have tags; both users and linklists have links. For each user and linklist, we can aggregate all tags in the links to represent the user and linklist. For user we can add the interest in the user profile to represent user. After we get the representation of users and linklists, we can use the content based filtering method for recommendation.

For links recommendation, the number of tags is not enough to represent the link. In this case, we have to use the collaborative filtering methods. There are memory based and model based two methods for collaborative filtering algorithms. However considering the user data in Whaam website is not enough to generate a model for recommendation and the complicity of the model-based algorithm, we will use memory based method in collaborative filtering algorithms.

3.2 Proposed Methodology

The details of recommender system algorithms are discussed in this section.

3.2.1 Tag Stemming

Stemming is the process for changing words to their stem or root form. Before analyzing the tag frequency, tags need to be stemming and combined. For example, if one user has saved five links with tag “travel” and another five links with tag “travelling”, they should be combined that the user saves ten links with tag “travel”. We will use a stemmer implemented by Martin Porter, because it is very widely used and it becomes the de facto standard algorithm used for English stemming [16].

3.2.2 Tags Frequency Times Inverse Users Frequency

By analyzing users’ tags, we can calculate tags frequency by each user, which means how many times user uses this tag, and inverse users
frequency, which represents the importance of the tag to user. The inverse users frequency is important because it eliminates the impact of the common tags. For example, if all users have one tag, this tag does not contribute to distinguish users from each other. The inverse users frequency in this case will be zero which will remove the tags.

\[
TF - IUF_{ui} = \frac{\text{freq}(\text{tag}_{ui})}{\max\{\text{freq}(\text{tags}_u)\}} \times \log \left( \frac{|\text{Users}|}{|\text{UsersTag}_{i}|} \right)
\]

- \(TF - IUF_{ui}\): tags frequency times inverse users frequency by user \(u\) and tag \(i\)
- \(\text{freq}(\text{tag}_{ui})\): the frequency of tag \(i\) by user \(u\)
- \(\max\{\text{freq}(\text{tags}_u)\}\): the maximum frequency of all tags by user \(u\)
- \(|\text{Users}|\): the number of all users
- \(|\text{UsersTag}_{i}|\): the number of users who have tag \(i\)

### 3.2.3 User’s Cosine Similarity

Cosine similarity is normally used to measure similarity between two vectors. After calculating the TF-IUF for each user’s tags, for each user, we can get a vector, which contains all the TF-IUF the user has. Then we can measure the two users’ similarity by applying cosine similarity on the two vectors.

\[
\text{UserSimilarity}_{u,v} = \frac{\sum_{i=1}^{n} \text{UserTag}_{ui} \times \text{UserTag}_{vi}}{\sqrt{\sum_{i=1}^{n} (\text{UserTag}_{ui})^2} \times \sqrt{\sum_{i=1}^{n} (\text{UserTag}_{vi})^2}}
\]

- \(\text{UserSimilarity}_{u,v}\): the cosine similarity between user \(u\) and \(v\)
- \(\text{UserTag}_{ui}\): user \(u\), tag \(i\) ’s TF-IUF

### 3.2.4 Linklist’s Similarity

Linklist is a group of links that are correlative by some kinds of meaning and saved together by a user in one place. When calculating the similarity of linklists, we can use the same method in 3.2.2 and 3.2.3. Instead of processing the user’s tags, we calculate the similarity by processing the linklist’s tags usage.

\[
TF - ILF_{ui} = \frac{\text{freq}_(\text{tag}_{ui})}{\max\{\text{freq}_(\text{tags}_{i})\}} \times \log \left( \frac{|\text{Linklists}|}{|\text{LinklistsTag}_{i}|} \right)
\]
• TF − ILF\(_{li}\): tags frequency times inverse linklists frequency by linklist \(l\) and tag \(i\)
• \(\text{freq}(\text{tag}_{li})\): the frequency of tag \(i\) by linklist \(l\)
• \(\max\{\text{freq}(\text{tags})\}\): the maximum frequency of all tags by linklist \(l\)
• \(|\text{Linklists}|\): the number of all linklists
• \(|\text{LinklistsTag}_{i}|\): the number of linklists which have tag \(i\)

\[
\text{LinklistSimilarity}_{lm} = \frac{\sum_{i=1}^{n} \text{LinklistTag}_{li} \times \text{LinklistTag}_{mi}}{\sqrt{\sum_{i=1}^{n} (\text{LinklistTag}_{li})^2} \times \sqrt{\sum_{i=1}^{n} (\text{LinklistTag}_{mi})^2}}
\]

• \(\text{LinklistSimilarity}_{lm}\): the cosine similarity between linklist \(l\) and \(m\)
• \(\text{LinklistTag}_{li}\): linklist \(l\), tag \(i\) ’s TF-ILF

3.2.5 Users Recommendation

For each user, we can recommend this user’s top-k most similar users, which the similarity is above predefined threshold \(k\), minor the users, which this user already follows.

\[\text{RecUser}_u = \{v | \text{similarity}(u, v) > k\} \setminus \{v | v \in \text{Following}_u\}\]

• \(\text{RecUser}_u\): is the recommended users for user \(u\)
• \(\text{similarity}(u, v)\) is the similarity between user \(u\) and \(v\)
• \(k\) is the threshold for similarity
• \(\text{Following}_u\): is the users which user \(u\) follows

Using this method, we can recommend users who share the same interests.

3.2.6 Linklists Recommendation

Linklists can be recommended by taking the top-\(k\) most similar linklists from results in 3.2.4.
RecLinklist\textsubscript{(u,m)} = \{l \mid \text{similarity}(l, m) > k\} \setminus \{l \mid l \in \text{Linklists}_u\}

- RecLinklist\textsubscript{(u,m)}: is the recommended linklists for user u based on linklist l
- similarity(l,m) is the similarity between linklist l and m
- k is the threshold for similarity
- Linklists\textsubscript{u}: is the user u’s linklists

### 3.2.7 Links Recommendation

For links recommendation, since each link only has a few tags or even no tags, the number of tags is not enough to use the similar tag based method, which we discussed in 3.2.5 or 3.2.6. We will use the collaborative filtering method for links recommendation. Links are normally saved to linklists so we can calculate the recommendation data by either users or linklists. The two different methods are explained below.

**User based links recommendation**

We use user-to-item based collaborative filtering to recommend links. There is no rating to links by user and user can only like or save links, so the rating vector is actually a binary vector. Then the cosine similarity of link l2 and l2 can be simplified as:

\[
\text{Similarity}_{(l1,l2)} = \frac{|U_{(l1,l2)}|}{\sqrt{|U_{l1}| \times |U_{l2}|}}
\]

- |U\textsubscript{(l1,l2)}|: the number of users who saved both link l1 and l2
- |U\textsubscript{l1}|: the number of users who saved link l1
- |U\textsubscript{l2}|: the number of users who saved link l2

**Linklist based links recommendation**

When users save links, they usually save them into different linklists. Linklists, which have the same link, can be the natural cluster of links for link l. So we can recommend links based on linklists using methods in 3.2.7:
\[
\text{Similarity}_{(l_1,l_2)} = \frac{|L_{(l_1,l_2)}|}{\sqrt{|L_{l_1}| \times \sqrt{|L_{l_2}|}}.
\]

- $|L_{(l_1,l_2)}|$: the number of linklists which have both link l1 and l2
- $|L_{l_1}|$: the number of linklists which have link l1
- $|L_{l_2}|$: the number of linklists which have link l2

### 3.2.8 Solving Sparsity Problem in Collaborative Filtering Method

In 3.2.7, we use the user-to-item based collaborative filtering to recommend links. However, if some links are only saved by a small number of users or to a small number of linklists and there are no common users or linklists. Then no links will be recommended. To solve this problem, we can use the linklists similarity data (3.2.4) to recommend similar links in the similar linklists cluster. For example, link l1 is only saved to linklist l1. From 3.2.4, we can get a list of linklists, which are similar to linklist l1 and above some predefined threshold. Then we aggregate all links in these linklists, calculate link frequencies, and recommend the top-k most frequent links to the user.

### 3.3 System Prototype Design and Implementation

#### 3.3.1 System Overview

The recommender system will be used as a service behind the Whaam web application. The system has two subsystems, one is the online and the other is offline. The arrow in below figure presents the data flow of different systems. The offline recommend system get input from the database and online recommend subsystem and output statistics to another persistent store. The online recommend system get input from database, offline recommend system and statistics data store and output recommend items to the Whaam web application.
3.3.2 Offline Data Processing Subsystem

The main function of the offline subsystem is processing the user data, handling real-time input from online subsystem and saving calculated statistics data to persistent store. The process of offline data processing subsystem is shown in Figure 3.3.

Figure 3.2: Whaaam recommender system overview

Figure 3.3: Offline recommend subsystem process
In general, the offline recommend subsystem is a daemon process which loops the data selection, data denormalization, tag stemming and recommend algorithms steps while also listens to live event from online recommend subsystem and aggregates the events to recent updates. There are five main components in Figure 3.3: Real time event handler, Data selection, Data denormalization, Tag stemming and Recommend algorithms. We will discuss the function of each component in below chapters.

3.3.2.1 Real Time Event Handler

Real time event handler subscribes all kinds of event from online recommend subsystem. All events, including creation events and update events, are aggregated and saved in the persistent store. Because the data is denormalized and saved redundantly, these events are necessary to make all the redundant data up to date. The persistent recent updates are also the input for the data selection process, so only the hot data’s recommendation is updated depends on the computation complexity. The recent updates record contains id, type (Link/Linklist/User) and timestamp (when event happens).

Besides the create/update events from website, the explicit input from users about recommended items are also recorded. For each recommended item, user can click it, like it and remove it as not relevant. All these inputs from user are saved in order to provide better recommendation next time.

3.3.2.2 Data Selection

The data selection component is the offline recommendation scheduler. It decides how many similarity pairs need to be recalculated in this round based on user/item popularity and times needed in this round. After similarity pairs and recommend candidates are updated, the timestamp is also saved.

New items and updated items are selected, and their owners are selected for candidates updates. For new users, there is no user data for them and they are recommended popular items instead of similar items, so they are selected and marked as new user. Users who updated their interest in profile are selected. All delete items and users are selected, because their indexes need to be removed. For privacy, no private saves and linklists are selected.

3.3.2.3 Data Denormalization
In this step, data dispersed in different tables is aggregated into one record. For example, tags will be aggregated together to one string separated by comma for links and linklists; links’ and linklists’ ids will be aggregated together for users. This step is important for the recommend algorithm step, because in relational database, data is normalized to different tables, denormalized data can improve the processing speed. Not all data is denormalized during this step and only the data, which is used during recommendation, is denormalized.

- **User Denormalization**
  The user entry will have all the tags used, all the links ids saved, all the linklists ids created and subscribed, all the links ids liked, all the linklists ids liked, all interests and all the followings ids.

- **Linklist Denormalization**
  The linklist entry will have all the tags used, all users ids that created and subscribed the linklist, all the links ids saved in the linklist.

- **Link Denormalization**
  The link entry will have all the tags used, all users ids that saved the link, all users ids that liked the link and all linklists ids that contains the link.

- **Tag denormalization**
  The tag entry will have all the users ids that used the tags and all the linklists that used this tag.

For each item and user, after data denormalization, the timestamp for this entry in the persistent store is updated.

### 3.3.2.4 Tag Stemming

All tags are stemming and merged for all links, linklists and users. First, tags are stemming to the root of the English words. Then the component goes through the stemmed tags and merges the tags counts, which have the same stemming root. For example, if a user have 5 sport tags and 5 sporting tags, after this step, the user will have 10 sport tags.

### 3.3.2.5 Recommend Algorithm

This is the most time-consuming step (Figure 3.4). During this step, the data similarity and recommend candidates are calculated. In
order to speed up this process, the intermediate data is also cached and a timestamp is generated when the data is calculated.

The similarity calculation will calculate similarity for each pair of items and user, which is not necessary for every round of this offline processing. In each round, only the recently updated items and users are calculated similarities. The input from data selection provides which data is needed to update in this round. The basic statistics are data from the data denormalized process.

![Figure 3.4: Recommend algorithm process](image)

**3.3.3 Online Recommend Service Subsystem**

The function of the online subsystem is handling request from web application, fetching data from statistics, populating with user data and returning results to the web application.
The online recommend subsystem is the interface between web application and the offline recommend subsystem. There are four main components in the subsystem Figure 3.5: Request handler, Candidates selection, Privacy control and Renderer. We will discuss the function of each component in below chapters.

### 3.3.3.1 Request Handler

Request handler is the interface between web application and online recommend service. Three kinds of requests are handled here.

<table>
<thead>
<tr>
<th>Request Type</th>
<th>Description and Handling Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creation/update events</td>
<td>When users/links/linklists are created or updated, the notification is received by the handler. All events will be forwarded to the offline subsystem to update statistics.</td>
</tr>
<tr>
<td>User explicit rating</td>
<td>When user explicitly votes the recommended results. The voting is saved in user preferences for further analysis.</td>
</tr>
<tr>
<td>Recommend request</td>
<td>When a recommend request is received, system goes to the next step to generate the recommendation.</td>
</tr>
</tbody>
</table>

Table 3.2: Types of request from web application and handling methods

### 3.3.3.2 Candidates Selection
In this step, recommended candidates, which are generated in the offline subsystem, are provided for recommendation. User preferences are also used to exclude explicitly rated items. Normally there are more candidates than what are returned to users. For simplicity, we use random strategy. A randomly chosen set is selected from the candidates.

### 3.3.3.3 Privacy Control

Before recommendation results are returned to user, privacy control is applied for linklist recommendation. In Whaam, linklist can have public, private and share-with-friends privacy. The private linklists should be filtered out before return the result to webserver. Private data has already been ignored during the offline processing part. However, due to data during the recommendation may not be consistent with current data, some linklists may be set to private during the offline processing process. These linklists needs to be filtered out.

### 3.3.3.4 Renderer

Here we use JSON to format the recommended result. The result has an array of recommended entries and a generated timestamp. For each recommended entry, only id and similarity value are returned. The web server can use the ids to retrieve contents and show the recommend item details to the user.

### 3.4 Prototype Implementation Details

In this section, we list the implementation details of the prototype, including used language and libraries, different real time event types, the data structure of statistics data in Redis and the client API usage.

#### 3.4.1 Language and Libraries Choices

The language and libraries we use to implement the recommender system prototype is as follows:

**Python** is a general-purpose, high-level programming language whose design philosophy emphasizes code readability [19]. Python is easy to use, and we will use Python as the main implementing language, so that we can focus the algorithms and function parts.

**NumPy** is the fundamental package for scientific computing library with Python [20]. We will use it to calculate vector production, mean value and other statistics operations.
Sqlalchemy is the python SQL toolkit and object relational mapper [21]. In Whaam, all data is stored in MySQL. We use Sqlalchemy to fetch the data from the database and denormalize it to Redis.

Flask is a microframework for Python based on Werkzeug [22]. We use it to implement the web service. The service handles request and serves JSON response, which is simple, so we don't need a full stack web framework like Django.

Redis is an advanced key-value store [23]. It can be used as a data structure server because keys can contain hashes, lists, sets and sorted sets besides strings. We use it to store the recommendation data and statistics data. We will explain the detailed data structure used in the prototype in Chapter 3.4.3.

Matplotlib is a python 2D plotting library [24]. We use it to generate histograms and bar charts during the evaluation. Since we use python as the main implementing language, it is easy to integrate a plotting library in Python to generate graph.

### 3.4.2 Realtime Events Implementation

Whaam uses an event system to dispatch events generated in the business logic layer. The generated event is sent to message queue service (RabbitMQ). For other services, they can subscribe the events they interest, so when event is received by the message queue, it is broadcasted to all subscribers. The request handler in online subsystem subscribes events to notify offline system to update index and statistics. The events, which are subscribed, are list in Table 3.3.

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Type</th>
<th>Action</th>
<th>Event Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>User</td>
<td>Create</td>
<td>New user registered event</td>
</tr>
<tr>
<td>Like/UnlikeLink</td>
<td>User</td>
<td>Like/Unlike</td>
<td>User likes or dislikes a link</td>
</tr>
<tr>
<td>Like/UnlikeLinkSave</td>
<td>Link</td>
<td>Like/Unlike</td>
<td>User likes or dislikes a link save, it can be combined with Like/UnlikeLink</td>
</tr>
<tr>
<td>SaveLink</td>
<td>Link</td>
<td>Like/Unlike</td>
<td>User saves a link</td>
</tr>
<tr>
<td>CreateLinklist</td>
<td>Linklist</td>
<td>Create</td>
<td>User creates a linklist</td>
</tr>
<tr>
<td>DeleteLink</td>
<td>Link</td>
<td>Delete</td>
<td>User deletes a link</td>
</tr>
<tr>
<td>DeleteLinklist</td>
<td>Linklist</td>
<td>Delete</td>
<td>User deletes a linklist</td>
</tr>
<tr>
<td>EditLink</td>
<td>Link</td>
<td>Update</td>
<td>User edits a link</td>
</tr>
<tr>
<td>EditLinklist</td>
<td>Linklist</td>
<td>Update</td>
<td>User edits a linklist</td>
</tr>
<tr>
<td>Follow/UnfollowUser</td>
<td>User</td>
<td>Follow/Unfollow</td>
<td>User follows or unfollows another user</td>
</tr>
<tr>
<td>Subscribe/UnsubscribeLinklist</td>
<td>Linklist</td>
<td>Subscribe/Unsubscribe</td>
<td>User subscribes or unsubscribes a linklist</td>
</tr>
</tbody>
</table>
Events are serialized using JSON format. An example of the event json message is:

```
{
    "eventType": "SaveLink",
    "user_id": 5,
    "data": {
        "id": 123,
        "title": "A sample link",
        "description": "This is a sample link",
        "tags": "Technology, Recommender System, Data Mining",
        ...
    },
    "created": "2013-02-12T14:23:10Z"
}
```

The message is parsed and saved in the Redis as recent update as:

```
{
    "type": "Link",
    "action": "Create",
    "data": {
        "user_id": 5,
        "id": 123
        "tags": "Technology, Recommender System, Data Mining"
    },
    "timestamp": "2013-02-12T14:23:10Z"
}
```

After events are received, based on event type, data is parsed and useful data is forwarded to the offline subsystem. The request handler component in the online subsystem provides an adapter for the offline subsystem, so the changes in the business logic event data will not impact the offline subsystem.

### 3.4.3 Recommendation Data Structure on Redis

Redis is used as a key-value store for the denormalized data and recommendation statistics. We use hashes, lists, sorted sets data structure for different purpose:

- **Recent Updates**
  We store recent updates events as a list under the key “recent_updates”. The online subsystem pushes event in the list as a producer while the offline subsystem pops event as a consumer.

- **Denormalized Data**
Denormalized data is stored under the key `<type>-<obj_id>` and value is stored as a hash. For example, for user 1, the key is user-1 and the value contains link_ids, following_ids, linklist_ids, tags and timestamp.

- **Similarities Data**
Similarities data is stored under the key similarity-<algorithm>-<type>-<obj_id> and value is stored as a sorted set. In the set, the object id, which is pair of the similarities of the object id in the key, is stored as member and the similarity is stored as score. In Redis, sorted set can return a range of members by score, so we can use it to retrieve the most similar objects for one object. For example, similarity-tags-linklist-4 is the key for tags based similarity data of linklist 4. Similarity-cf-user-5 is the key for collaborative filtering similarity data of user 5.

3.4.4 Client API
The client API uses REST over HTTP and the response is formatted as JSON. There are two kinds of recommendation requests: one is similar object request and the other is the recommended object for the user. The input and output of the two kinds of requests are:

Request 10 links which are similar to target link with id 15:
http://recommend.whaam.com/similar?type=link&target_id=15&limit=10
```
{
  "data": [
    {
      "id": 104,
      "score": 0.99
    },
    {
      "id": 134,
      "score": 0.79
    },
    ...
  ],
  "timestamp": "2013-03-25T15:03:24Z",
  "timetogenerate": "200ms"
}
```

Request 10 users, which are recommended to user with id 5:
http://recommend.whaam.com/recommend?type=user&user_id=5&limit=10
```
{
  "data": [
    {
      "id": 67,
      "score": 0.99
    },
    {
      "id": 68,
      "score": 0.89
....
```

The limit parameter can limit the number of recommended objects. The result is sorted by the score of each object. The score is the similarities of the object with the target object.
4. Evaluation

In this section, we first give the evaluation methodology. Then we discuss how we set up the experiments and show the experiments results.

4.1 Evaluation Methodology

We use offline evaluation for the recommender system. The mean average precision is calculated during the offline evaluation.

4.1.1 Offline Evaluation

Offline evaluation is used to test our algorithm. In offline evaluation, no actual users are involved and we use current existing data set. The data set is split in a sample and test set. The sample set is used to calculating the recommendation and the test set is used to evaluate the recommendation result.

4.1.2 Precision and Recall

Precision and recall are measurement for evaluation from information retrieval field, and they are used in recommender system evaluation as well [17]. For any recommended items, they can be relevant to user’s interest or not relevant.

Precision is defined as the fraction of recommended items that are relevant to user’s interest.

\[
\text{Precision} = \frac{|\{\text{Recommended Items}\} \cap \{\text{Relevant Items}\}|}{|\{\text{Recommended Items}\}|}
\]

Recall is defined as the fraction of relevant items that are recommended.

\[
\text{Recall} = \frac{|\{\text{Recommended Items}\} \cap \{\text{Relevant Items}\}|}{|\{\text{Relevant Items}\}|}
\]

4.1.3 F-Measure and Mean Average Precision

Precision and recall are observed inversely related, which means when the number of recommended items increases, the recall will increase while precision decreases. F-measure takes account both
precision and recall and produce a single metric [17]. The balanced F-measure’s equation is as follows:

\[ F = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \]

Another way to measure recommender system is Mean average precision, which means value of average precision. Average precision also considers the order of the recommended items. It calculates precision and recall at every position of the ranked sequence of recommended items and gets the average value. For example, if we recommend n items to users and the relevant items is m, the equation to compute average precision is [18]:

\[ \text{AP}_n = \sum_{k=1}^{n} \frac{P(k)}{\min(m, n)} \]

- \text{AP}_n is the average precision for recommend n items to user
- \text{P}(k) is the precision at cut-off k in the recommended item list

We will use mean average precision to measure the accuracy of our recommender system algorithm since average precision considers the order of the recommended items and the mean value evaluates the overall precision of the recommender system.

4.2 Experiment Setup

In this section, we first list the data used for evaluation and then the experiment processes.

4.2.1 Data setup

The offline processing data is collected from Whaam database. The statistics of the dataset is in Table 2.

<table>
<thead>
<tr>
<th>Number of users</th>
<th>~1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of links</td>
<td>~15000</td>
</tr>
<tr>
<td>Number of linklists</td>
<td>~1700</td>
</tr>
<tr>
<td>Number of tags</td>
<td>~13000</td>
</tr>
<tr>
<td>Number of likes</td>
<td>~3000</td>
</tr>
</tbody>
</table>
Table 4.1: the statistics of the dataset in the evaluation

| Number of subscribes | ~700 |

4.2.2 Experiment Process

We will use the offline recommend subsystem prototype to run the evaluation. For users recommendation, what we want is user follows that we recommended. Therefore, we can use the follower/following data as the test data set to measure the recommendation result. For links and linklists recommendation, the sample data is selected from the whole data set and input to the offline recommend subsystem. Then the recommend result is evaluated with the test data set.

4.3 Experiment Results

In this section, we show the experiment results for users recommendation, linklists recommendation and links recommendation.

4.3.1 Users Recommendation

User normally follows other users who share the same interests. For users recommendation evaluation, we can compute the average precision of the users which user follows comparing to the users that we recommend. Then for each user, we calculate the average precision to get the mean average precision. The evaluation result is in Figure 4.1.

![Figure 4.1: Mean average precision of recommending 5, 10 or 15 users](image)
The mean average precision of tag based cosine similarity recommendation is higher than the traditional collaborative filtering method. We can also see the result the recommendation methods are much better than the simple random method. In Figure 4.1, we can also see that the MAP decreases when the number to recommend increases.

The mean average precision is the mean value of average precision of the all the users and the average precision for each user varies from 0.0 to 1.0. We can have the distribution of the average precision for all users figure. For example, when recommend 10 users, the distribution using tag based algorithm is in Figure 4.2 and the distribution using collaborative filtering method is in Figure 4.3.

![Figure 4.2](image1.png)  
Figure 4.2 Distribution of tag based average precision of all users when recommend 10

![Figure 4.3](image2.png)  
Figure 4.3 Distribution of collaborative filtering average precision of all users when recommend 10

From the distribution figures, we can see the average precision varies between different users. Some users have zero precision while some
other users even have almost 1.0 precision. One common characteristic of these users is that they do not have much data, so the precision is either too high or too low. The recommendation algorithms are based on the existing data of users, so if there is not enough data, the recommendation is not accurate. To evaluate the overall performance of the recommendation algorithms, we will only consider the mean average precision of all users.

### 4.3.2 Linklists Recommendation

Users like linklists, which are similar to their own linklists. For each linklist of one user, we can recommend several similar linklists to the user. Then we compare the recommended linklists with user's other liked or subscribed linklists to calculate the average precision. Then we can get the mean average precision for this user's all linklists. We do this calculation to all the users and calculate the mean average precision for all the users and all the linklists. The mean average precision result is shown in Figure 4.4.

![Figure 4.4: MAP value of recommending 5, 10 or 15 linklists](image)

In Figure 4.4, we can see the similar result to the user recommendation. The tag based cosine similarity method is better than the traditional collaborative filtering method, which is better than the random method.

### 4.3.3 Links Recommendation
For links recommendation, since links only have several tags that are not enough to apply the tag based method, so we use the collaborative filtering method based on linklist. During the experiment, we calculate the mean average precision both using the linklist based collaborative filtering method and user based linklist. The random generated links are also shown for comparison.

![Chart](chart.jpg)

**Figure 4.5: Mean average precision value of recommending 5, 10 or 15 links**

From the result in Figure 4.5, we can see that the mean average precision of linklist based collaborative filtering recommendation is higher than the user based collaborative filtering method. The mean average precision decreases as the number to recommend increases.
5. Conclusion

5.1 Discussion

From the evaluation result, we can see that tag based recommendation algorithm is better than the traditional collaborative filtering method. This result is predictable because of the properties of the dataset we chose. The dataset is collected from Whaam, a social bookmark service, in which there are more items like links and linklists than users. Each user only likes or saves a small number of items which makes collaborative filtering method worse than the tag based method. The tag-based recommendation is a kind of content-based method. In content-based method, the main problem is to find a representation of the item, which makes the content easy to analyze. In the Whaam dataset, tags are the natural representation for links. By calculating the tags frequency, it also becomes the representation for users and linklists. In the tags based, there is no sparsity problem, which is in collaborative filtering method. For example, both of the two users like songs by Michael Jackson. One user saves the link of the song “black and white”, while the other user saves the link of the song “dangerous”. If collaborative filtering method is used, songs will not be recommended to each other since these two songs are different. However, if both of the two songs have tags “song” and “Michael Jackson”, for example, the tag-based method will recommend songs with same tags to each other.

Another finding in the evaluation result is for links recommendation; the linklist based collaborative filtering method is better than the user based collaborative filtering method. User groups their links into linklists based on categorization. The links in one linklist are more correlated than all links of this user. For example, some users may like both music and travel; another user likes travel and technology. If we use the user based collaborative filtering method, it is possible that a music link is recommended to the user who likes travel and technology. However, if we use the linklist based collaborative filtering method, only the links that have similar meanings will be recommended.

The tag based method and linklist based collaborative filtering method are useful under the two assumptions. One is that most users will tag links and most tags are correct. The other one is that most
users will put links that are correlated into linklists not put them randomly. If that isn’t the case, we need to think of methods to create the two pre-conditions before we use the two algorithms. For example, if some users randomly add or tag links, they need to be filtered before making recommendation. For the links without tags, we need to generate the tags from the links content and let user to modify them later.

Lacking of data can impact the precision of the recommender system, so users have to input enough data before we recommend data to users. In the distribution figures 4.2 and 4.3, we can see some users have very low and very high precision compared to the mean average precision of all users. For these users, we can set a threshold that if users do not have enough data; we recommend popular data to users instead.

About the scalability of the prototype, the two subsystems are needed to scale together. For the online subsystem, we can scale it by adding more servers and distributing user’s requests to different servers and the algorithm in the online subsystem does not need to be changed. However, for the offline subsystem, as more and more data is added, it will become the bottleneck of the system. In that case, we need to change the recommendation algorithm to use distributed computing method. For example, we can use map-reduce and Hadoop to compute the statistics of the dataset instead of computing on one machine using Redis to store the statistics.

5.2 Future Work

The future improvement of the recommender system based on the work conducted can be divided into three areas: recommendation algorithm improvement, scalability and multi-language support.

In our work, we use tags to represent links. About pre-processing the tags, we only use the stemming method but this does not work for the synonymic tags. Combining the synonymic tags for links, users and linklists can improve the recommendation precision. Another improvement is for links without tags, and then we need to auto-generate tags from title, description etc. contents for these links. Besides the behavior pattern of users can be analyzed and the spam users should be filtered before generating recommendation result. This can improve the user satisfaction. The recommendation method we use only considers the users’ interests without time. Users’
interests may changes from time to time. Users’ current interests should have higher priority than the old interests. Current popular items should have higher priority than the old items. By considering the time factor, we can reorder the recommended list and give most relevant links to users.

For the scalability problem, the offline recommendation data processing will be the bottleneck. As more and more links are added to the system, one server is not enough to compute and store all the data. A cluster of servers will work together to calculate the similarity of links, users and linklists. When something changes and we need to recalculate the similarity, if the size of data is extremely large, it is not feasible to calculate for each pair. The data can be divided into many small clusters, so that only a cluster of data needs to be recalculated.

Multi-language support is not considered in current work. If there is multi-language content in the social bookmark system, the recommender system needs to make sure only links under user’s language preference are recommended. A language detector can be added to support this function.

5.3 Conclusion

In this study, we analyze the data structure and relations in Whaam and propose suitable algorithms for the recommender system, which can be used in social bookmark service like Whaam. We use different algorithms for different recommendations. For users and linklists recommendation, a tag based cosine similarity method is used. First, tags frequency inverse tag usages by all users or linklists are calculated for each user or linklist. Then for each pair, cosine similarity is calculated based on tags frequency. The recommendation is based on top-k most similar users or linklists. For links recommendation, a linklist based collaborative filtering method is used. For each pair of link, the link similarity is calculated by the ratio of how many common linklists that have both of the links to the total amount of linklists by either one of the links.

A prototype of recommender system based on social bookmark for discovery engine is designed and implemented. The system can recommend users to users, linklists to users and links to users. The recommender system has two subsystems: online subsystem and
offline subsystem. The online subsystem fetches the recommendation candidates from offline subsystem, filters them and responds to user’s request on real time. The offline subsystem calculates the data statistics and similarity. It listens to live event from online subsystem and aggregates the events to recent update. A daemon process loops the data selection, data denormalization, tag stemming and recommend algorithms steps to update recommendation candidates.

The experiments show that our recommender system based on social bookmark using tag based cosine similarity method improves the mean average precision by 45% compared to traditional collaborative filtering method for users and linklists recommendation. For link recommendation, the linklist based collaborative filtering method increase the mean average precision by 39% compared to the user based collaborative filtering method.
References


